CLARIN and Free Open Source Finite-State Tools

Kimmo Koskenniemi and Anssi Yli-Jyrä
September 2008

University of Helsinki
Department of General Linguistics
CLARIN: Common Language Resources and Technology Infrastructure Network (1)

- Plenty of language resources (i.e. materials and tools) exist and could be used in many combinations, but because they are fragmented it is difficult:
  - to obtain a permission to use them and
  - to find out whether useful materials and tools exist,
  - to combine the materials and tools, because they may not be compatible with each other.

- CLARIN aims at an environment where materials and tools can be located, accessed and used seamlessly and therefore, has to solve the above problems.
CLARIN: … (2)

- CLARIN is one of the 34 European infrastructures on the [ESFRI](#) (European Strategic Forum for Research Infrastructures) roadmap.
- CLARIN is in its [preparatory phase](#), i.e. building the rules, norms, legal frameworks and agreements.
- CLARIN has [EU Commission funding](#) for 2008-2010
  - Coordinator Steven Krauwer, Utrecht
  - Executive Board (Work Package leaders)
  - 32 partners receiving (at least some) EU funding (4.1 M €)
  - >100 member organizations
- [National infrastructure funding](#) supplements CLARIN (>>EU funding)
CLARIN: … (3)

- CLARIN is for humanities, including the study of languages, but also historical studies, literature, etc.
- One vision is that all European heritage of written and recorded speech would be available on the desktop of researchers, enabling new ways to combine evidence.
- CLARIN is multidisciplinary, and many sciences, including computer science and language technology will cooperate in building and exploiting it.
Multilinguality in CLARIN (1)

- There are some **100 languages with a significant national status in Europe** (including 23 official EU languages) and many more dialects and earlier forms.
- CLARIN needs to provide a **platform** for all these languages (and for many more important and immigrant languages).
- Materials in various languages need to be processed, searched and used in **combinations**.
- Specifically, the CLARIN software needs to **handle all languages in a manageable way**, e.g. to perform multilingual searches. (Cf. problem 3 on 1st slide)
Multilinguality in CLARIN … (2)

- BLARK (Basic Language Resource Kit) specifies sets of language materials, language modules and applications necessary for a language to be used in modern society.
- Only a few of the 100 languages have satisfactory BLARK resources yet. (They will be built in CLARIN construction phase.)
- Most of the existing language modules are technically incompatible with each other (each has a software module of its own).
- Finite-state techniques could provide a technically uniform platform capable of processing any language for CLARIN.
Finite-state techniques and CLARIN

- Finite-state techniques can be used for morphological analysis and for phrase-level processing (cf. also NooJ which does syntactic parsing).
- Finite-state transducers (FSTs) and automata (FSAs) are technically language independent – just transition networks.
- The use of FSAs and FSTs is simple to implement for runtime software.
- What we need are
  - compilers to compile language specific descriptions into FSTs or FSAs and
  - language specific rules, lexicons and descriptions.
Existing finite-state software

- Lots of finite-state packages have been created by different researchers and teams (maybe 50-100), see e.g. https://kitwiki.csc.fi/twiki/bin/view/KitWiki/FsmReg.
- Some are free open source (OpenFST, SFST, Vaucanson), others commercial (XFST, ATT FSM) or with other restrictions (e.g. shareware).
- Some are actively maintained and developed, many have neither maintenance nor other activities.
Using FST software

- Robust free implementations of FST technology exist but they differ somewhat from each other:
  - names of functions, their parameters, data types and concepts (open vs. closed alphabet; input/ upper and output/ lower tapes, etc.) differ.
- Usually the documentation is defective and the programmer has to consult the implementation code in order to find out the details.
- The programmer has to commit to one system before he can write FST tools (and changing will be painful).
HFST: Helsinki Finite State Toolkit

- Does not implement yet another finite-state calculus, but rather utilizes existing free open source implementations:
  - SFST (Helmut Schmid) transducers without weights,
  - OpenFST (M. Riley, J. Schalkwyk, W. Skut, C. Allauzen and M. Mohri) with weighted transducers, and others.
- Intends to provide practical and general purpose free open source finite-state tools.
- Provides a carefully defined and well documented common interface to several FSM engines.
- Implements useful tools on top of this HFST interface.
- Builds some full scale language modules for testing the HFST tools and the interface.
HFST team

- Krister Lindén, PhD, leader of the team
- Anssi Yli-Jyrä, PhD, researcher providing theory and compilation formulas
- Erik Axelsson, programming the HFST interface
- Miikka Silfverberg, programming the HFST-TWOC rule compiler
- Tommi Pirinen, MA, programming HFST-LEXC lexicon compiler and a free Finnish morphological analyzer
- Kimmo Koskenniemi, professor, funding and consulting
Goals of the HFST

- Create *convergence and cooperation* within the community which develops finite-state calculus and tools.
- Create a *neutral platform* where different implementations of the FS calculus can *coexist and compete with each other*.
- Create a *critical mass of research* for improving the basic algorithms of the calculus, and compilation algorithms.
- Stimulate the production of free open source *software for compiling* dictionaries, grammars and rules into FSTs.
- Stimulate the production of language *resources* (e.g. dictionaries, grammars, rules) *to be compiled into FSTs*.
- Inspired by *CLARIN infrastructure* and supports it.
First tools to be implemented with HFST

- **Lexicon compilers**: HFST-LEXC (like Xerox LEXC) useful for complex morphological phenomena such as agglutination, derivation and compounding. Derivation and compounding can be cyclic.

- **Morphophonemic rule compilers**: HFST-TWOLC: Two-level compiler (like Xerox TWOLC); later on also a compiler for replace operations (like SFST and Xerox XFST)

- **Test cases**: Finnish has lots of word-forms (> $10^{24}$) and a lot of morphophonological alternations (consonant gradation, vowel harmony, stem final alternations. Northern Sámi is even more complex.
Words are composed out of morphemes. Morphemes correspond to entries in the lexicon.

Morphemes are grouped into classes with similar distribution: morphemes belong to the same class, if they can be interchanged (regarding what precedes). Classes are represented as sublexicons.

Morphemes may still differ regarding what classes of morphemes may follow, i.e. there is a continuation class (which points to a sublexicon from which the next morpheme can be chosen).

The lexicon compiler reads in a lexicon and produces a FST out of it.
An example of a lexicon

LEXICON Root
talo N;
kaTu N;
LEXICON N;
+N+Sg+Nom:0 #;
+N+Sg+Gen:n #: 
+N+Sg+Ine:ssA #
+N+Pl+Gen:jen #;
Root;
n Root;
END
HFST-TWOLC: Two-level rule compiler

- Morphemes are considered to have two representations
  - lemma and features to be shown as the result of the analysis
  - a morphophonemetic representation of the morpheme
- The task for the rule component is to relate the morphophonemetic representation to the surface form, e.g.
  \[\text{pettää} +V +Pss +Pcp2 +Sup +Pl +Ine\]
  \[\text{pettÄtTUImPissA}\]
  \[\text{pet0et0yimmissä}\]
- Rules are parallel with no rule ordering, e.g.
  \[T:0 \leftrightarrow t \_ \text{Vowel} (I:i) \text{Cons} (\text{Cons} \mid :\#)\]
  \[I:j \leftrightarrow :\text{Vowel} \_ :\text{Vowel}\]
  \[A:a \leftrightarrow (:a \mid :o \mid u:) :* \_ \text{(elsewhere ä)}\]
The SFST structure

- regular expression formalism
- translation into function calls
- functions for the finite-state calculus
Design of the HFST

- Implementation of HFST-SFST regular expression formalism
- Implementation of HFST-XFST regular expression formalism
- Implementation of HFST-TWOLC rule compiler
- Implementation of HFST-LEXC lexicon compiler

HFST interface

- SFST finite-state calculus
- OpenFST finite-state calculus
- Vaucanson finite-state calculus

… etc …
HFST interface

- Definitions of underlying concepts and functions.
- Consistent naming of functions and use of parameters.
- C++ code as needed to achieve similar behaviour when using different calculus packages.
- DOXYGEN documentation of functions and parameters: www.ling.helsinki.fi/kieliteknologia/tutkimus/hfst/
- TWiki user manuals, technical details and development ideas.
- File formats (from AT&T, OpenFST, SFST) for exchanging binary and text mode FSTs.
HFST and formalisms

- HFST is not a new formalism
- HFST is an attempt to standardize the interface between formalisms and implementation class libraries
- HFST is intended for implementations of
  - old formalisms such as TWOLC, LEXC, SFST-PL
    (needed to secure the future of the existing resources)
  - new programming languages such as Kleene
- The HFST effort includes substantial original research on compilation methods needed close to the HFST interface
  - The Generalized Restriction (GR) (YJ&K 2004) operation and its optimized implementations and extensions
  - TWOL and rewriting rules with GR (YJK 2006, YJ 2007a)
  - Supporting also GR-related predicate logics (e.g. Hulden 2008)
Rule compilation in HFST-based formalisms (1)

- Key insight: universal quantifiers over substrings $x$
  - YJK04,06,YJ07a: **general restriction** (GR) operator $=^M>:$
    $W =^M> W' \overset{def}{=} \neg \Pi_M(W \cap \neg W')$ where $W,W \subseteq \Sigma^*(M\Sigma^*)^{\leq k}$,
    $M \cap \Sigma = \emptyset$, and $\Pi_M$ removes markers $M$
    - used to express $\forall vxy$ where $v,x,y$ are substrings (YJK04,06)
    - GRs can be nested with regular ops and itself (YJ07a)
  - $\forall x$ of Hulden's (08) **predicate logic** reduces to a GR
    - $(\forall x)(\varphi) \overset{def}{=} \neg \Pi_{(x)}((\Sigma^*(x)\Sigma^*(x)\Sigma^*) \cap \neg (\varphi)) = (W =^M> \varphi)$

- Relationship with GR to FO logic over strings is well known:
  - GR used in FSIG (YJK 04,YJ 05a) and TWOL (YJK 06)
    - The uses were known to be FO definable (YJ03,05a,05b)
  - dot-depth hierarchy $= \forall \exists$-alternation hier. (Thomas 82)
    - $D$-$d$ of FSIG & nested GR discussed by YJ (05a,05b,07a)
  - Hulden (08) uses a duplicated **named** marker $(x)$ that is drawn from $M$ to indicate start/end of substring $x$ (cf.YJK04,06,YJ07a)
Rule compilation in HFST-based formalisms (2)

- Before GR: e.g. Kaplan and Kay (KK): rewrite rules to FSTs.
  - Koskenniemi modified the comp.method for two-level rules
- GR captures all generalized two-level rules (YJK 04, 06):
  - e.g. $\Sigma^*(t:d)\Sigma^* = (\Sigma^*a(\Sigma^*a\Sigma^*)$ says that $t:d$ may only occur between two a's
  - center $t:d$ generalizes to languages of substrings
    - was formerly approximated in XFST (Karttunen 2004)
    - exact + elegant solutions through GR or the related logic (YJK 04, Karttunen 2004, YJK 06, 07a-c, Hulden 08)
- GR in parallel, directed, ranked rewriting (YJ07a,07b,07c)
- GRs closed under conjunctions of GRs (YJK06,YJ07a,b):
  - $(V =^M > V') \cap (W =^M > W') \overset{\text{def}}{=} (x)V(x) \cup (y)W(y) =^M > ((x)V'(x) \cup (y)W'(y))$
  - TWOL conflict resolution with coherent intersection:
    - $(V =^M > V') \cap (W =^M > W') \overset{\text{def}}{=} (V \cup W) =^M > ((V \cap V') \cup (W \cap W'))$
Rule compilation references

  --: *Contributions to the Theory of Finite-State Based Grammars*. Dissertation. (2005a)
  --: *Linguistic Grammars with Very Low Complexity*. In Arppe et al., Inquiries into Words, Constraints and Contexts. CSLI Publications. (2005b)
  --: *Transducers from Parallel Replace Rules and Modes with Generalized Lenient Composition*. FSMNLP 2007 postproceedings. (2007b)
HFST and existing parsers and other software

- FSTs produced with HFST tools could be reused in existing parsers, e.g. morphological analysis of languages with complex alternations and inflection.
- This might simplify the structure of e.g. syntactic parsers.
- Heuristic methods for collecting named entities and new lexemes could be based on HFST tools and reused.
- Dictionaries created in other formalisms could be reused with the HFST tools.
- Some internal functionalities in other parsers could be optimized or generalized using the HFST API and modules.
OMor: open source morphologies

- For research, we need free open source resources which we can improve and develop as needed. In parallel with HFST, we have OMor projects for creating free morphological analyzers.
- OMorFi creates a Finnish transducer lexicon based on a list of words of a dictionary with inflectional codes.
- A project in Norway has created two-level morphological analyzers for Northern and Lule Sámi using Xerox tools. These will be reimplemented with HFST tools as open source.
- HFST demos already exist for Finnish (OMorFi), Swedish, English and French.
HFST commercial and open source applications

- HFST is written under GNU LGPL, SFST is GPL, OpenFST is under Apache license.
  - These are all free open source licenses.
  - Any further tools developed from HFST or combined with it will remain under the strictest license of the combined software.
  - Later: the Apache license for the code of the HFST proper
- Any FSTs produced with HFST tools will remain under the same conditions as the input lexicon and rules, i.e. either proprietary commercial or free open source.
- Thus, HFST tools can be used both for open source and proprietary projects.
A note on finite-state patents

- Xerox and AT&T have claimed several patents on finite-state techniques and their applications (maybe 100).
- If they are "software patents", they are not in effect in Europe. (Some may be and others perhaps not.)
- Many patents are invalid because there is previous commercial use of the method or the method is obvious and the only solution. Invalidating patents is expensive. (Patents are a bit like a minefields.)
- Patents do not prevent private or "fair use".
- Most use of HFST (and other finite-state tools) is, thus, safe. The use of FSTs produced this way should also be safe.